FUNTOOLS: FITS USERS NEED TOOLS FOR QUICK, QUANTITATIVE ANALYSIS

NASA Grant NAG5-9484

Final Report

For the Period 1 May 2000 through 30 April 2001

Principal Investigator
Eric Mandel

June 2001

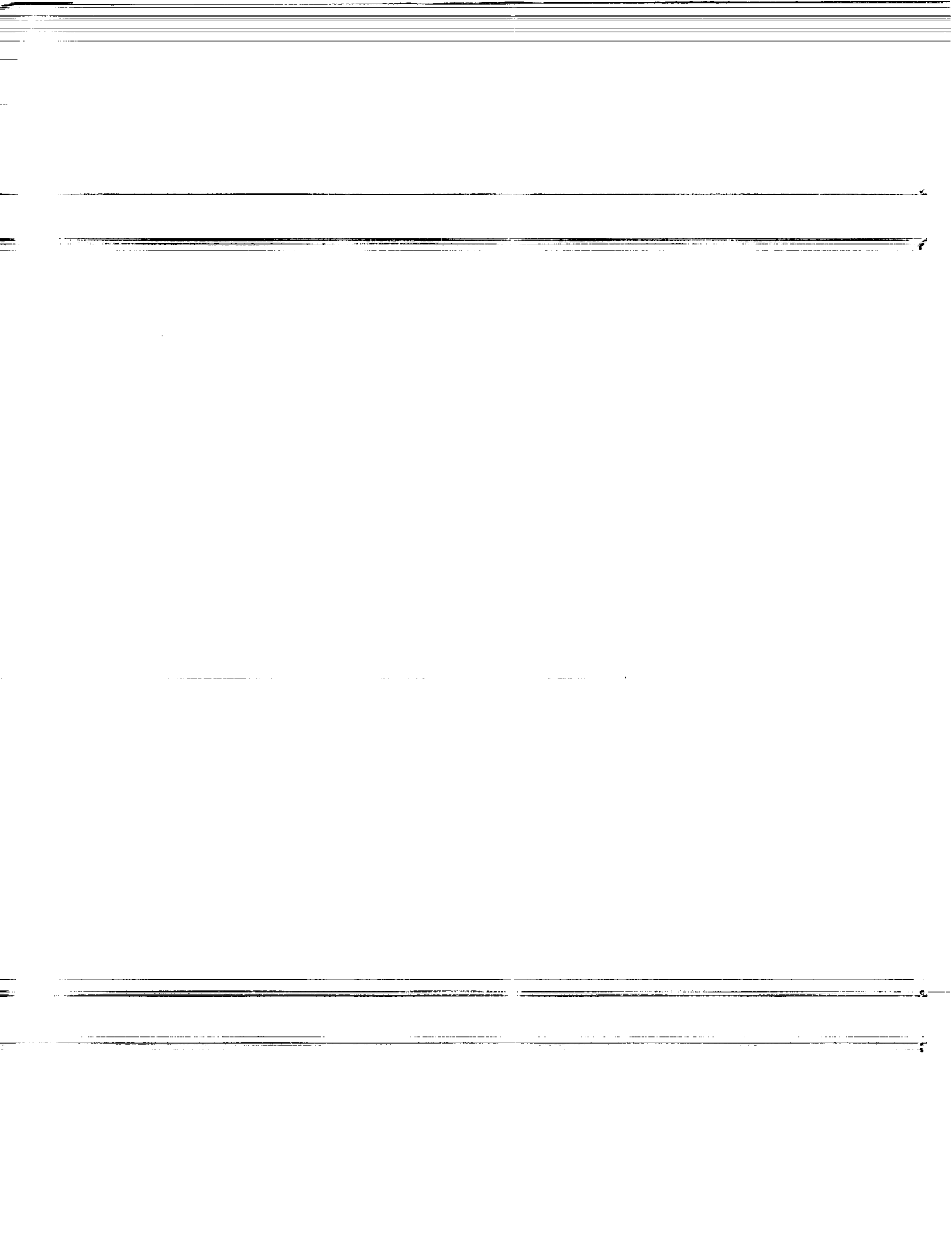
Prepared for:

National Aeronautics and Space Administration
Washington, D.C. 20546

Smithsonian Institution
Astrophysical Observatory
Cambridge, Massachusetts 02138

The Smithsonian Astrophysical Observatory
is a member of the
Harvard-Smithsonian Center for Astrophysics

The NASA Technical Officer for this grant is Joe Brederkamp, Code 077.0, NASA Headquarters, Washington, DC 20546-0001.



1 Introduction

The Funtools project arose out of conversations with astronomers about the decline in their software development efforts over the past decade. A stated reason for this decline is that it takes too much effort to master one of the existing FITS libraries simply in order to write a few analysis programs. This problem is exacerbated by the fact that astronomers typically develop new programs only occasionally, and the long interval between coding efforts often necessitates re-learning the FITS interfaces.

We therefore set ourselves the goal of developing a minimal buy-in FITS library for researchers who are occasional (but serious) coders. In this case, "minimal buy-in" meant "easy to learn, easy to use, and easy to re-learn next month". Based on conversations with astronomers interested in writing code, we concluded that this goal could be achieved by emphasizing two essential capabilities. The first was the ability to write FITS programs without knowing much about FITS, *i.e.*, without having to deal with the arcane rules for generating a properly formatted FITS file. The second was to support the use of already-familiar C/Unix facilities, especially C structs and Unix stdio. Taken together, these two capabilities would allow researchers to leverage their existing programming expertise while minimizing the need to learn new and complex coding rules.

For example, our group at SAO pursues research in X-ray astronomy, where the primary data are stored as rows of photon events in FITS binary tables. Each row consists of information about a single photon event, *e.g.*, position, energy, and arrival time. In order to process photons in an X-ray table, one typically reads the events from stdin into an array of C records, manipulates these event records, and then writes the results to stdout:

```
while( (nev=fread(ebuf, sizeof(EvRec), MAXEV, stdin)) ){
  for(i=0; i<nev; i++){
    ev = ebuf + i;
    if( ev->pha == XXX ) ev->energy = NewPHA(ev->pha);
  }
  fwrite(ebuf, sizeof(EvRec), nev, stdout);
}
```

In essence, the goal of our Funtools effort was to implement this code fragment as simply as possible for FITS binary tables.

2 Design Approach

Our design approach was to *minimize the number of public subroutines* in the Funtools library. This approach was based on the hypothesis that a few carefully crafted routines would be easy to learn, easy to remember, and easy to use by occasional coders. In other words, we sought to make it easier to "see the forest for the trees" by minimizing the number of trees.

The implications of this approach were two-fold. Firstly, the routines would have to be useful in their basic operation, while containing appropriate hooks to activate more sophisticated functionality. That is, we tried to avoid automatically adding routines to the library in order to extend functionality. Instead we sought to accomplish this aim by adding optional keyword specifiers to the calling sequence of our basic routines, taking care to avoid making those routines overly complex along the way.

Secondly, we decided that routines should act on behalf of the coder, especially with regard to FITS formatting issues (*e.g.*, header generation and extension padding). We recognized that proper formatting of FITS files is one of the most difficult and tedious parts of FITS programming, and we wanted Funtools to provide this service automatically. For example, when image or binary table data is written, the associated headers should be generated automatically as needed. In doing so, great care must be taken to ensure that all necessary parameters (including user-specified parameters) are correctly incorporated into the header. Designing such automatic services required that we maintain the "state" of processing for a given FITS file, so that the library could "do the right thing" on behalf of the user.

In considering these design principles, we concluded that it would be necessary to impose a few rules of "natural order" on coders. Of course, order is important for any I/O library: one cannot read a file before it is opened or after it is closed. But in the current case, we decided to make this concept explicit. For example, in order to ensure that headers are automatically and properly generated when an image is written to a FITS extension using stdio, we imposed the rule that all parameters must be written before outputting the image data itself. We hoped that users would accept such rules if they resulted in services being performed automatically and correctly. Examples of "natural order" are discussed below.

3 Implementation

Our implementation strategy was centered on perfecting the code needed by a select few image and table processing algorithms, including the canonical X-ray analysis task: for each X-ray event (row in a binary table), read selected input columns into user space, modify the value of one or more of these columns, and output the results by merging new value(s) with the original input columns. The Funtools library implements this standard X-ray event-processing algorithm in a simple and straightforward manner:

```
typedef struct eventrec{ int pha, energy; } *Ev, EvRec;

ifun = FunOpen("in.fit[EV+,pha=5:9||pha==pi]", "rc", NULL);
ofun = FunOpen("out.fits", "w",  ifun);

FunColumnSelect(ifun, sizeof(EvRec), "merge=update",
   "pha",    "J", "r",  FUN_OFFSET(Ev, pha),
   "energy", "J", "rw", FUN_OFFSET(Ev, energy), NULL);

while( (ebuf=(Ev)FunTableRowGet(ifun, NULL, MAXEV, NULL, &nev)) ){
  for(i=0; i<nev; i++){
    ev = ebuf + i;
    if( ev->pha == XXX ) ev->energy = NewPHA(ev->pha);
  }
  FunTableRowPut(ofun, ebuf, nev, 0, NULL); free(ebuf);
}
FunClose(ofun); FunClose(ifun);
```

The Funopen() routine opens the specified FITS image or binary table extension and sets up the required column and spatial region filters. As shown above, the "output" call can be passed an input handle (argument 3) to inherit parameters and other information from the input file. In addition, if the input file is opened in "copy" mode ("c" in argument 2), the user can automatically copy the input extensions to the output file by appending "+" to the extension name ("EV+" in the example). The "natural order" rule here is that the input file must be opened before the output file.

The heart of Funtools processing for binary tables (*i.e.*, X-ray events) is the FunColumnSelect() routine, which specifies how to read columns into a user-defined C struct and/or how to write columns to an output file. Named columns (arguments 4, 8, *etc.*) are read into the record structure offsets (arguments 7, 11, *etc.*) They are automatically converted from the FITS data type to the user-specified data type (arguments 5, 9, *etc.*) In addition, if the output file has inherited the input handle, then the output file also will know about columns having "w" mode. Finally, the optional "merge=[update/replace/append]" string (argument 3) specifies that processed columns will be merged with the original input columns on output. Thus, the same subroutine can be used to set up reading, writing, and/or merging of named columns.

Once columns from the binary table have been selected for processing, the routines FunTableRowGet() and FunTableRowPut() are used to get and put rows. (Similar routines are available for image processing.) FunTableRowGet() reads event rows and stores the selected columns into an array of structs. Buffer space is allocated on the fly if argument 2 is NULL. FunTableRowPut() writes the event rows to the output file. Note that FITS headers are generated and written automatically as needed. To support these automatic services, the natural order rule is that get and put calls must be alternated.

Finally, the FunClose() routine is called to flush and close Funtools files. Note that FITS extension padding will be added automatically as needed (although one can call FunFlush() explicitly). Also note that the remaining input extensions are copied automatically to the output if in "copy" mode. To support these services, the natural order rule is to close the output file before the input file(s) to copy remaining extensions.

In Funtools, much of the complexity of dealing with FITS is hidden. FITS headers and extension padding are written automatically as needed. Output files easily inherit input parameters and other extension information. Copy of input extensions is specified easily on the command line. The price paid for these automatic services is the imposition of some rules of "natural order," although options are available for cases where coders are forced to violate these rules. For example, if input files must be closed before output files, coders can call FunFlush() beforehand to copy the remaining input files explicitly.

4 Advanced Filtering Capabilities

As part of the Funtools library, events in FITS Binary tables and raw event files can be filtered or selected based on a flexible set of user-specified criteria that allows column values to be compared with numeric values, header parameters, functions, or other column values.

To filter events (and spatial regions, as described below), the filter specification is added to the FITS filename using bracket notation:

```
foo.fits[pha==1&&pi==2&&circle(512,512,10)]
```

The filter specification comes after the extension and image sections and may optionally be enclosed in its own set of brackets. Thus:

```
foo.fits[EVENTS,400:599,400:599,pha==1&&pi==2&&circle(512,512,10)]
```

and

```
foo.fits[BKGD][400:599,400:599][circle(512,512,10)]
```

define filters on an image section of size 200x200 defined on the FITS EVENTS (binary table) and BKGD (image) extensions.

In addition to filtering columns of binary tables, our funtools library also can filter both events and images using spatial regions specifiers. Spatial region filtering allows a program to select regions of an image or event list to process using simple geometric shapes and boolean combinations of shapes. When an image is filtered, only pixels found within these shapes are processed. When an event list is filtered, only events found within these shapes are processed.

Spatial region filtering for images and events is accomplished by means of region specifications. A region specification consists of one or more region expressions, which are geometric shapes,combined according to the rules of boolean algebra. Region specifications also can contain comments and local/global processing directives.

Typically, region specifications are specified using bracket notation appended to the filename of the data being processed:

```
foo.fits[circle(512,512,100)]
```

It is also possible to put region specification inside a file and then pass the filename in bracket notation:

```
foo.fits[@my.reg]
```

A region descriptor consists of one or more region expressions or regions, separated by comas, new-lines, or semi-colons. Each region consists of one or more geometric shapes combined using standard boolean operation. Several types of shapes are supported, including:

```
shape: arguments:
----- ---------
ANNULUS xcenter ycenter inner_radius outer_radius (outer_radius) ...
ANNULUS xcenter ycenter inner_radius outer_radius (n=<number>)
BOX xcenter ycenter xwidth yheight (optional angle)
CIRCLE xcenter ycenter radius
ELLIPSE xcenter ycenter xwidth yheight angle
FIELD none
LINE x1 y1 x2 y2
PANDA xcenter ycenter angle1 angle2 nangle iradius oradius nradius
PIE xcenter ycenter angle1 angle2 (angle3) (angle4) (angle5) ...
PIE xcenter ycenter angle1 angle2 (n=<number>)
POINT x1 y1
POINT x1 y1 x2 y2 ... xn yn
POLYGON x1 y1 x2 y2 ... xn yn
```

In addition, region shapes can be combined together using Boolean operators:

```
Symbol Operation Use
-------- --------- ---

  ! or !! not Exclude this shape from this region
  & or && and Include only the overlap of these shapes
  | or || inclusive or Include all of both shapes
  ^   exclusive or Include both shapes except their overlap
```

Also each region has a coordinate system used that is used to interpret the region, i.e., to set the context in which position and size values are interpreted. For this purpose, the following keywords are recognized:

```
PHYSICAL # pixel coords of original file using LTM/LTV
IMAGE # pixel coords of current file
FK4, B1950 # sky coordinate systems
FK5, J2000 # sky coordinate systems
GALACTIC # sky coordinate systems
ECLIPTIC # sky coordinate systems
ICRS # currently same as J2000
LINEAR # linear wcs as defined in file
AMPLIFIER # mosaic coords of original file using ATM/ATV
DETECTOR # mosaic coords of original file using DTM/DTV
```

To implement event and spatial region filtering, funtools utilizes a technique in which the filter specification is converted into a small C program, which is compiled and linked automatically so that events or image sections can be fed to this program and filter results returned to the calling program. The power of the technique lies in these considerations:

- The generated filter program is very small, containing approximately 200 lines of code, so that it compiles and links in about second or less on most modern machines.
- The filter specification itself becomes a hard-wired part of this program, so that filter checking is performed as part of compiled code, not in the usual interpreted mode. This use of a compiled filter results in a speed-up factor of 4-5 over previous techniques, even after the program compilation time is added.
- All C syntax and C operators become valid parts of the filter syntax, making available a much wider range of filter possibilities than previously.

5 Dissemination

A paper discussing Funtools entitled "Funtools: an Experiment in Minimal Buy-in Software" (E. Mandel, S.S. Murray, J. Roll) was presented at the ADASS200 conference.

Funtools is available in beta release at http://hea-www.harvard.edu/RD/funtools. The library has been ported to Sun/Solaris, Linux, Dec Alpha, SGI, and Windows (using Cygwin). We also offer a number of sample programs such as funcnts, which calculates the background-subtracted image counts in user-specified regions. A public release of the Funtools package is scheduled for June 2001.